Pushdown compression

Pilar Albert, Elvira Mayordomo, Philippe Moser, Sylvain Perifel

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 39-48
www.stacs-conf.org

PUSHDOWN COMPRESSION

P. ALBERT [1], E. MAYORDOMO [1], P. MOSER [2], AND S. PERIFEL [3]

[1] Dept. de Informática e Ingeniería de Sistemas , Universidad de Zaragoza. Edificio Ada Byron, María de Luna 1 - E-50018 Zaragoza (Spain)
E-mail address: {mpalbert,elvira}@unizar.es

[2] Dept. of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland
E-mail address: pmoser@cs.nuim.ie

[3] LIP, École Normale Supérieure de Lyon. UMR 5668 ENS Lyon, CNRS, UCBL, INRIA
E-mail address: asylvain.perifel@ens-lyon.fr

Abstract. The pressing need for efficient compression schemes for XML documents has recently been focused on stack computation [6, 9], and in particular calls for a formulation of information-lossless stack or pushdown compressors that allows a formal analysis of their performance and a more ambitious use of the stack in XML compression, where so far it is mainly connected to parsing mechanisms. In this paper we introduce the model of pushdown compressor, based on pushdown transducers that compute a single injective function while keeping the widest generality regarding stack computation.

The celebrated Lempel-Ziv algorithm LZ78 [10] was introduced as a general purpose compression algorithm that outperforms finite-state compressors on all sequences. We compare the performance of the Lempel-Ziv algorithm with that of the pushdown compressors, or compression algorithms that can be implemented with a pushdown transducer. This comparison is made without any a priori assumption on the data's source and considering the asymptotic compression ratio for infinite sequences. We prove that Lempel-Ziv is incomparable with pushdown compressors.

1. Introduction

The celebrated result of Lempel and Ziv [10] that their algorithm is asymptotically better than any finite-state compressor is one of the major theoretical justifications of this widely used algorithm. However, until recently the natural extension of finite-state to pushdown compressors has received much less attention, a situation that has changed due to new specialized compressors.

Key words and phrases: Finite-state compression, Lempel-Ziv algorithm, pumping-lemma, pushdown compression, XML document.

Research supported in part by Spanish Government MEC Project TIN 2005-08832-C03-02 and by Aragón Government Dept. Ciencia, Tecnología y Universidad, subvención destinada a la formación de personal investigador-B068/2006.

In particular, XML is rapidly becoming a standard for the creation and parsing of documents, however, a significant disadvantage is document size, even more since present day XML databases are massive. Since 1999 the design of new compression schemes for XML is an active area where the use of syntax directed compression is specially adequate, that is, compression centered on the grammar-based generation of XML-texts and performed with stack memory [6, 9].

On the other hand the work done on stack transducers has been basic and very connected to parsing mechanisms. Transducers were initially considered by Ginsburg and Rose in [4] for language generation, further corrected in [5], and summarized in [1]. For these models the role of nondeterminism is specially useful in the concept of λ-rule, that is a transition in which a symbol is popped from the stack without reading any input symbol.

In this paper we introduce the concept of pushdown compressor as the most general stack transducer that is compatible with information-lossless compression. We allow the full power of λ-rules while having a deterministic (unambiguous) model. The existence of endmarkers is discussed, since it allows the compressor to move away from mere prefix extension by exploiting λ-rules.

The widely-used Lempel-Ziv algorithm LZ78 [10] was introduced as a general purpose compression algorithm that outperforms finite-sate compressors on all sequences when considering the asymptotic compression ratio. This means that for infinite sequences, the algorithm attains the (a posteriori) finite state or block entropy. If we consider an ergodic source, the Lempel-Ziv compression coincides exactly with the entropy of the source with high probability on finite inputs. This second result is useful when the data source is known, whereas it is not informative for arbitrary inputs. We don't know the performance of Lempel-Ziv on individual long or infinite inputs (notice that an infinite sequence is Lempel-Ziv incompressible with probability one). For the comparison of compression algorithms on general sequences, either an experimental or a formal approach is needed, such as that used in [8]. In this paper we follow [8] using a worse case approach, that is, we consider asymptotic performance on every infinite sequence.

We compare the performance of the Lempel-Ziv algorithm with that of the pushdown-compressors, or compression algorithms that can be implemented with a pushdown transducer. This comparison is made without any a priori assumption on the data's source and considering the asymptotic compression ratio for infinite sequences.

We prove that Lempel-Ziv compresses optimally a sequence that no pushdown transducer compresses at all, that is, the Lempel-Ziv and pushdown compression ratios of this sequence are 0 and 1, respectively. For this result, we develop a powerful nontrivial pumping-lemma, that has independent interest since it deals with families of pushdown transducers, while known pumping-lemmas are restricted to recognizing devices [1].

In fact, Lempel-Ziv and pushdown compressing algorithms are incomparable, since we construct a sequence that is very close to being Lempel-Ziv incompressible while the pushdown compression ratio is at most one half. While Lempel-Ziv is universal for finite-state compressors, our theorem implies a strong non-universality result for Lempel-Ziv and pushdown compressors.

The paper is organized as follows. Section 2 contains some preliminaries. In section 3, we present our model of pushdown compressor with its basic properties and notation. In section 4 we show that there is a sequence on which Lempel-Ziv outperforms pushdown compressors and in section 5 we show that Lempel-Ziv and pushdown compression are incomparable. We

finish with a brief discussion of connections and consequences of these results for dimension and prediction algorithms.

2. Preliminaries

We write $\mathbb{Z}$ for the set of all integers, $\mathbb{N}$ for the set of all nonnegative integers and $\mathbb{Z}^+$ for the set of all positive integers. Let Σ be a finite alphabet, with $|\Sigma| \geq 2$. Σ^* denotes the set of finite strings, and Σ^∞ the set of infinite sequences. We write $|w|$ for the length of a string w in Σ^*. The empty string is denoted by λ. For $S \in \Sigma^\infty$ and $i, j \in \mathbb{N}$, we write $S[i..j]$ for the string consisting of the i^{th} through j^{th} bits of S, with the convention that $S[i..j] = \lambda$ if $i > j$, and $S[0]$ is the leftmost bit of S. We write $S[i]$ for $S[i..i]$ (the i^{th} bit of S). For $w \in \Sigma^*$ and $S \in \Sigma^\infty$, we write $w \sqsubseteq S$ if w is a prefix of S, i.e., if $w = S[0..|w| - 1]$. Unless otherwise specified, logarithms are taken in base $|\Sigma|$. For a string x, x^{-1} denotes x written in reverse order. We use $f(x) = \perp$ to denote that function f is undefined on x.

Let us give a brief description of the Lempel-Ziv (LZ) algorithm [10]. Given an input $x \in \Sigma^*$, LZ parses x in different phrases x_i, i.e., $x = x_1 x_2 \dots x_n$ ($x_i \in \Sigma^*$) such that every prefix $y \sqsubset x_i$, appears before x_i in the parsing (i.e. there exists $j < i$ s.t. $x_j = y$). Therefore for every i, $x_i = x_{l(i)} b_i$ for $l(i) < i$ and $b_i \in \Sigma$. We sometimes denote the number of phrases in the parsing of x as $P(x)$. After step i of the algorithm, the i first phrases $x_1, \dots, x_i$ have been parsed and stored in what we will call the *dictionary*. Thus, each step adds one word to the dictionary.

LZ encodes x_i by a prefix free encoding of $x_{l(i)}$ and the symbol b_i, that is, if $x = x_1 x_2 \dots x_n$ as before, the output of LZ on input x is

$$LZ(x) = c_{l(1)} b_1 c_{l(2)} b_2 \dots c_{l(n)} b_n$$

where c_i is a prefix-free coding of i (and $x_0 = \lambda$).

LZ is usually restricted to the binary alphabet, but the description above is valid for any Σ.

For a sequence $S \in \Sigma^\infty$, the LZ infinitely often compression ratio is given by

$$\rho_{LZ}(S) = \liminf_{n \to \infty} \frac{|LZ(S[0 \dots n-1])|}{n \log_2(|\Sigma|)}.$$

$\rho_{LZ}(S)$ corresponds to the best-case performance of Lempel-Ziv on finite prefixes of sequence S.

We also consider the almost everywhere compression ratio

$$R_{LZ}(S) = \limsup_{n \to \infty} \frac{|LZ(S[0 \dots n-1])|}{n \log_2(|\Sigma|)}.$$

$R_{LZ}(S)$ corresponds to the worst-case performance of Lempel-Ziv on finite prefixes of sequence S.

3. Pushdown compression

Definition 3.1. A *pushdown compressor* (PDC) is a 7-tuple

$$C = (Q, \Sigma, \Gamma, \delta, \nu, q_0, z_0)$$

where

- Σ is the finite input alphabet

- Q is a finite set of states
- Γ is the finite stack alphabet
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to Q \times \Gamma^*$ is the transition function
- $\nu : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \Sigma^*$ is the output function
- $q_0 \in Q$ is the initial state
- $z_0 \in \Gamma$ is the start stack symbol

We use δ_Q and δ_{Γ^*} for the projections of function δ. Note that the transition function δ accepts λ as an input character in addition to elements of Σ, which means that C has the option of not reading an input character while altering the stack. In this case $\delta(q, \lambda, a) = (q', \lambda)$, that is, we pop the top symbol of the stack. To enforce determinism, we require that at least one of the following hold for all $q \in Q$ and $a \in \Gamma$:

- $\delta(q, \lambda, a) = \perp$
- $\delta(q, b, a) = \perp$ for all $b \in \Sigma$

We restrict δ so that z_0 cannot be removed from the stack bottom, that is, for every $q \in Q$, $b \in \Sigma \cup \{\lambda\}$, either $\delta(q, b, z_0) = \perp$, or $\delta(q, b, z_0) = (q', vz_0)$, where $q' \in Q$ and $v \in \Gamma^*$.

There are several natural variants for the model of pushdown transducer [1], both allowing different degrees of nondeterminism and computing partial (multi)functions by requiring final state or empty stack termination conditions. Our purpose is to compute a total and well-defined (single valued) function in order to consider general-purpose, information-lossless compressors.

Notice that we have not required here or in what follows that the computation should be invertible by another pushdown transducer, which is a natural requirement for practical compression schemes. Nevertheless the unambiguity condition of a single computation per input gives as a natural upper bound on invertibility.

We first consider the transition function δ as having inputs in $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma^+$, meaning that only the top symbol of the stack is relevant. Then we use the extended transition function $\delta^* : Q \times \Sigma^* \times \Gamma^+ \to Q \times \Gamma^*$, defined recursively as follows. For $q \in Q$, $v \in \Gamma^+$, $w \in \Sigma^*$, and $b \in \Sigma$

$$\delta^*(q, \lambda, v) = \begin{cases} \delta^*(\delta_Q(q, \lambda, v), \lambda, \delta_{\Gamma^*}(q, \lambda, v)), & \text{if } \delta(q, \lambda, v) \neq \perp; \\ (q, v), & \text{otherwise.} \end{cases}$$

$$\delta^*(q, wb, v) = \begin{cases} \delta^*(\delta_Q(\delta^*_Q(q, w, v), b, \delta^*_{\Gamma^*}(q, w, v)), \lambda, \delta_{\Gamma^*}(\delta^*_Q(q, w, v), b, \delta^*_{\Gamma^*}(q, w, v))), \\ \qquad \text{if } \delta^*(q, w, v) \neq \perp \text{ and } \delta(\delta^*_Q(q, w, v), b, \delta^*_{\Gamma^*}(q, w, v)) \neq \perp; \\ \perp, \quad \text{otherwise.} \end{cases}$$

That is, λ-rules are inside the definition of δ^*. We abbreviate δ^* to δ, and $\delta(q_0, w, z_0)$ to $\delta(w)$. We define the *output* from state q on input $w \in \Sigma^*$ with $z \in \Gamma^*$ on the top of the stack by the recursion $\nu(q, \lambda, z) = \lambda$,

$$\nu(q, wb, z) = \nu(q, w, z)\nu(\delta_Q(q, w, z), b, \delta_{\Gamma^*}(q, w, z)).$$

The *output* of the compressor C on input $w \in \Sigma^*$ is the string $C(w) = \nu(q_0, w, z_0)$.

The input of an information-lossless compressor can be reconstructed from the output and the final state reached on that input.

Definition 3.2. A PDC $C = (Q, \Sigma, \Gamma, \delta, \nu, q_0, z_0)$ is *information-lossless* (IL) if the function

$$\Sigma^* \to \Sigma^* \times Q$$
$$w \to (C(w), \delta_Q(w))$$

is one-to-one. An *information-lossless pushdown compressor* ($ILPDC$) is a PDC that is IL.

Intuitively, a PDC *compresses* a string w if $|C(w)|$ is significantly less than $|w|$. Of course, if C is IL, then not all strings can be compressed. Our interest here is in the degree (if any) to which the prefixes of a given sequence $S \in \Sigma^\infty$ can be compressed by an ILPDC.

Definition 3.3. If C is a PDC and $S \in \Sigma^\infty$, then the *compression ratio* of C on S is

$$\rho_C(S) = \liminf_{n\to\infty} \frac{|C(S[0..n-1])|}{n \log_2(|\Sigma|)}$$

Definition 3.4. The *pushdown compression ratio* of a sequence $S \in \Sigma^\infty$ is

$$\rho_{PD}(S) = \inf\{\rho_C(S) \mid \text{ C is an ILPDC}\}$$

$\rho_{PD}(S)$ corresponds to the best-case performance of PD-compressors on S.

We can consider dual concepts R_C and R_{PD} by replacing lim inf with lim sup in the previous definition. $R_{PD}(S)$ corresponds to the worst-case performance of PD-compressors on S.

3.1. Endmarkers and pushdown compression

Two possibilities occur when dealing with transducers on finite words: should the end of the input be marked with a particular symbol # or not? As we will see, this is a rather subtle question. First remark that both approaches are natural: on the one hand, usual finite state or pushdown *acceptors* are one-way and do not know (and do not need to know) when they reach the end of the word; on the other hand, two-way finite state acceptors need to know it and everyday compression algorithms usually know (or at least are able to know) where the end of the input file takes place. For a word w, we will denote by $C(w)$ the output of a transducer C without endmarker, and $C(w\#)$ the output with an endmarker.

Unlike acceptors, transducers can take advantage of an endmarker: they can indeed output more symbols when they reach the end of the input word if it is marked with a particular symbol. This is therefore a more general model of transducers which, in particular, does not have the strong restriction of prefix extension: if there is no endmarker and C is a transducer, then for all words w_1, w_2, $w_1 \sqsubseteq w_2 \Rightarrow C(w_1) \sqsubseteq C(w_2)$. Let us see how this restriction limits the compression ratio.

Lemma 3.5. *Let C be an IL pushdown compressor with k states and working with no endmarker. Then on every word w of size $|w| \geq k$, the compression ratio of C is*

$$\frac{|C(w)|}{|w|} \geq \frac{1}{2k}.$$

Proof. Due to the injectivity condition, we can show that C has to output at least one symbol every k input symbols. Suppose on the contrary that there are words t, u, with $|u| = k$, such that C does not output any symbol when reading u on input $w = tu$. Then all the $k+1$ words t and $tu[0..i]$ for $0 \leq i \leq k-1$ have the same output by C, and furthermore two of them have the same final state because there are only k states. This contradicts injectivity. Thus C must output at least one symbol every k symbols, which proves the lemma. □

This limitation does not occur with endmarkers, as the following lemma shows.

Lemma 3.6. *For every k, there exists an IL pushdown compressor C with k states, working with endmarkers, such that the compression ratio of C on 0^n tends to $1/k^2$ when n tends to infinity, that is,*

$$\lim_{n\to\infty} \frac{|C(0^n)|}{n} = \frac{1}{k^2}.$$

Proof. (Sketch) On input 0^n, our compressor outputs (roughly) $0^{n/k^2}$ as follows: by selecting one symbol out of each k of the input word (counting modulo k thanks to k states), it pushes $0^{n/k}$ on the stack. Then at the end of the word, it pops the stack and outputs one symbol every k pop. Thus the output is $0^{n/k^2}$ (in fact, the remainder of n modulo k^2 also has to be taken into account).

To ensure injectivity, if the input word w is not of the form 0^n (that is, if it contains a 1), then C outputs $1w$. □

It is worth noticing that it is the injectivity condition that makes this computation impossible without endmarkers, because one cannot decide *a priori* whether the input word contains a 1. Thus pushdown compressors with endmarkers do not have the limitation of Lemma 3.5. Still, as Corollary 4.5 will show, pushdown compressors with endmarkers are not universal for finite state compressors, in the sense that a single pushdown compressor cannot be as good as any finite state compressor.

It is open whether pushdown compressors with endmarkers are strictly better than without, in the sense of the following question.

Open question. Do there exist an infinite sequence S, a constant $0 < \alpha \leq 1$ and an IL pushdown compressor C working with endmarkers, such that $\rho_C(S) < \alpha$, but $\rho_{C'}(S) \geq \alpha$, for every C' IL pushdown compressor working without endmarkers?

In the rest of the paper we consider both variants of compression: with and without endmarkers. We use the weakest variant for positive results and the strongest for negative ones, therefore showing stronger separations.

4. Lempel-Ziv outperforms Pushdown transducers

In this section we show the existence of an infinite sequence $S \in \{0,1\}^\infty$ compressible by Lempel-Ziv but not by pushdown compressors. More precisely, we show in Theorem 4.8 the following result: the almost everywhere Lempel-Ziv compression ratio on S is 0 but the infinitely often IL pushdown compression ratio is 1. Another (weaker) version will be stated in Theorem 4.9 for pushdown compressors with endmarkers.

The rough idea is that Lempel-Ziv compresses repetitions very well, whereas, if the repeated word is well chosen, pushdown compressors perform very poorly. We first show the claim on Lempel-Ziv and then prove a pumping-lemma for pushdown transducers in order to deal with the case of pushdown compressors.

4.1. Lempel-Ziv on periodic inputs

The sequence we will build consists of regions where the same pattern is repeated several times. This ensures that Lempel-Ziv algorithm compresses the sequence, as shown by the following lemmas.

We begin with finite words: Lempel-Ziv compresses well words of the form tu^n. The idea is that the dictionary remains small during the execution of the algorithm because there are few different subwords of same length in tu^n due to the period of size $|u|$. The statement is slightly more elaborated because we want to use it in the proof of Theorem 4.2 where we will need to consider the execution of Lempel-Ziv on a possibly nonempty dictionary.

Lemma 4.1. *Let $n \in \mathbb{N}$ and let $t, u, \in \Sigma^*$, where $u \neq \lambda$. Define $l = 1 + |t| + |u|$ and $w_n = tu^n$. Consider the execution of Lempel-Ziv on w_n starting from a dictionary containing $d \geq 0$ phrases. Then we have that*

$$\frac{|LZ(w_n)|}{|w_n|} \leq \frac{\sqrt{2l|w_n|}\log(d + \sqrt{2l|w_n|})}{|w_n|}.$$

This leads us to the following lemma on a particular infinite sequence.

Theorem 4.2 (LZ compressibility of repetitive sequences)**.** *Let $(t_i)_{i\geq 1}$ and $(u_i)_{i\geq 1}$ be sequences of words, where $u_i \neq \lambda, \forall i \geq 1$. Let $(n_i)_{i\geq 1}$ be a sequence of integers. Let S be the sequence defined by*

$$S = t_1 u_1^{n_1} t_2 u_2^{n_2} t_3 u_3^{n_3} \dots$$

If the sequence $(n_i)_{i\geq 1}$ grows sufficiently fast, then

$$R_{LZ}(S) = 0.$$

4.2. Pumping-lemma for injective pushdown transducers

This section is devoted to the statement and proof of a pumping-lemma for pushdown transducers. In the usual setting of recognition of formal languages by pushdown automata, the pumping-lemma comes from the equivalence between context-free grammars and pushdown automata, see for instance [11]. However, the proof is much less straightforward without grammars, as is our case since we deal with transducers and not acceptors. Moreover, there are three further difficulties: first, we have to consider what happens at the end of the word, after the endmarker (where the transducer can still output symbols when emptying the stack); second, we need a lowerbound on the size of the pumping part, that is, we need to pump on a sufficiently large part of the word; third, we need the lemma for an arbitrary finite family of automata, and not only one automaton. All this makes the statement and the proof much more involved than in the usual language-recognition framework. The proof consists in finding two similar configurations of the transducer so that we can repeat the input word read between them. The size of the input word has therefore to be large enough but note that in the following statement, this restriction is replaced by the possibility of pumping on an empty word u (as soon as $\alpha|w|^\beta < 1$ since we take the integer part).

Lemma 4.3 (Pumping-lemma)**.** *Let $\mathcal{F}$ be a finite family of ILPDC. There exist two constants $\alpha, \beta > 0$ such that $\forall w$, there exist $t, u, v \in \Sigma^*$ such that $w = tuv$ satisfying:*

- *$|u| \geq \lfloor \alpha |w|^\beta \rfloor$;*
- *$\forall C \in \mathcal{F}$, there exist two words x, y such that $C(tu^n) = xy^n$, $\forall n \in \mathbb{N}$.*

Taking into account endmarkers, we obtain the following corollary:

Corollary 4.4 (Pumping-lemma with endmarkers)**.** *Let $\mathcal{F}$ be a finite family of ILPDC. There exist two constants $\alpha, \beta > 0$ such that every word w can be cut in three pieces $w = tuv$ satisfying:*

(1) *$|u| \geq \lfloor \alpha |w|^\beta \rfloor$;*
(2) *there is an integer $c \geq 0$ such that for all $C \in \mathcal{F}$, there exist five words x, x', y, y', z such that for all $n \geq c$, $C(tu^n v\#) = xy^n z y'^{n-c} x'$.*

Let us state an immediate corollary concerning universality: pushdown compressors, even with endmarkers, cannot be universal for finite state compressors, in the sense that the compression ratio of a particular pushdown compressor cannot be always better than the compression ratio of every finite state compressor.

Corollary 4.5. *Let C be an IL pushdown compressor (with endmarkers). Then $\rho_C(0^\infty) > 0$. In particular, no pushdown compressor is universal for finite state compressors.*

Proof. By Corollary 4.4, there exist two integers k, k', ($k' \geq 1$), a constant $c \geq 0$ and five words x, x', y, y', z such that for all $n \geq c$, $C(0^k 0^{k'n} \#) = xy^n z y'^{n-c} x'$. By injectivity of C, y and y' cannot be both empty. Hence the size of the compression of $0^k 0^{k'n}$ is linear in n. This proves the first assertion.

Since for every $\epsilon > 0$ there exists an IL finite state compressor C' such that $R_{C'}(0^\infty) < \epsilon$, the pushdown compressor C cannot be universal for finite state compressors. □

4.3. A pushdown incompressible sequence

We now show that some sequences with repetitions cannot be compressed by pushdown compressors. We start by analyzing the performance of PDC on the factors of a Kolmogorov-random word (that is, a word w that contains at least $|w|$ bits of information in the (plain) Kolmogorov complexity sense, i.e. $K(w) \geq |w|$; or, to put it another way, a word that cannot be compressed by Turing machines). This result is valid even with endmarkers.

Lemma 4.6. *For every $\mathcal{F}$ finite family of $ILPDC$ with k states and for every constant $\epsilon > 0$, there exists $M_{\mathcal{F},\epsilon} \in \mathbb{N}$ such that, for any Kolmogorov random word $w = tu$, if $|u| \geq M_{\mathcal{F},\epsilon} \log |w|$ then the compression ratio for $C \in \mathcal{F}$ of u on input w is*

$$\frac{|C(tu)| - |C(t)|}{|u|} \geq 1 - \epsilon.$$

We can now build an infinite sequence of the form required in Theorem 4.2 that cannot be compressed by bounded pushdown automata. The idea of the proof is as follows: by Corollary 4.4, in any word w we can repeat a big part u of w while ensuring that the behaviour of the transducer on every copy of u is the same. If u is not compressible, the output will be of size almost $|u|$, therefore with a large number of repetitions the compression ratio is almost 1.

Theorem 4.7 (A pushdown incompressible repetitive sequence)**.** *Let Σ be a finite alphabet. There exist sequences of words $(t_k)_{k\geq 1}$ and $(u_k)_{k\geq 1}$, where $u_k \neq \lambda, \forall k \geq 1$, such that for every sequence of integers $(n_k)_{k\geq 1}$ growing sufficiently fast, the infinite string S defined by*

$$S = t_1 u_1^{n_1} t_2 u_2^{n_2} t_3 u_3^{n_3} \dots$$

verifies that

$$\rho_C(S) = 1,$$

$\forall C \in ILPDC$ (without endmarkers).

Combining it with Theorem 4.2 we obtain the main result of this section, there are sequences that Lempel-Ziv compresses optimally on almost every prefix, whereas no pushdown compresses them at all, even on infinitely many prefixes (Theorem 4.8) or using endmarkers (Theorem 4.9).

Theorem 4.8. *There exists a sequence S such that*

$$R_{LZ}(S) = 0$$

and

$$\rho_C(S) = 1$$

for any $C \in ILPDC$ (without endmarkers).

The situation with endmarkers is slightly more complicated, but using Corollary 4.4 (the pumping lemma with endmarkers) and a similar construction as Theorem 4.7 we obtain the following result. Note that we now use the limsup of the compression ratio for ILPDC with endmarkers.

Theorem 4.9. *There exists a sequence S such that*

$$R_{LZ}(S) = 0$$

and

$$R_C(S) = 1$$

for any $C \in ILPDC$ (using endmarkers).

5. Lempel-Ziv is not universal for Pushdown compressors

It is well known that LZ [10] yields a lower bound on the finite-state compression of a sequence [10], ie, LZ is universal for finite-state compressors.

The following result shows that this is not true for pushdown compression, in a strong sense: we construct a sequence S that is infinitely often incompressible by LZ, but that has almost everywhere pushdown compression ratio less than $\frac{1}{2}$.

Theorem 5.1. *For every $m \in \mathbb{N}$, there is a sequence $S \in \{0,1\}^\infty$ such that*

$$\rho_{LZ}(S) > 1 - \frac{1}{m}$$

and

$$R_{PD}(S) \leq \frac{1}{2}.$$

6. Conclusion

The equivalence of compression ratio, effective dimension, and log-loss unpredictability has been explored in different settings [2, 7, 13]. It is known that for the cases of finite-state, polynomial-space, recursive, and constructive resource-bounds, natural definitions of compression and dimension coincide, both in the case of infinitely often compression, related to effective versions of Hausdorff dimension, and that of almost everywhere compression, matched with packing dimension. The general matter of transformation of compressors in predictors and vice versa is widely studied [14].

In this paper we have done a complete comparison of pushdown compression and LZ-compression. It is straightforward to construct a prediction algorithm based on Lempel-Ziv compressor that uses similar computing resources, and it is clear that finite-state compression is always at least pushdown compression. This leaves us with the natural open question

of whether each pushdown compressor can be transformed into a pushdown prediction algorithm, for which the log-loss unpredictability coincides with the compression ratio of the initial compressor, that is, whether the natural concept of pushdown dimension defined in [3] coincides with pushdown compressibility. A positive answer would get pushdown computation closer to finite-state devices, and a negative one would make it closer to polynomial-time algorithms, for which the answer is likely to be negative [12].

Acknowledgement

The authors thank Victor Poupet for his help on the proof of Lemma 4.3.

References

[1] J. Autebert, J. Berstel, and L. Boasson. Context-free languages and pushdown automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, volume 1, Word, Language, Grammar*, pages 111–174. Springer-Verlag, 1997.

[2] J. J. Dai, J. I. Lathrop, J. H. Lutz, and E. Mayordomo. Finite-state dimension. *Theoretical Computer Science*, 310:1–33, 2004.

[3] D. Doty and J. Nichols. Pushdown dimension. *Theoretical Computer Science*, 381(1-3):105–123, 2007.

[4] S. Ginsburg and G. F. Rose. Preservation of languages by transducers. *Information and Control*, 9(2):153–176, 1966.

[5] S. Ginsburg and G. F. Rose. A note on preservation of languages by transducers. *Information and Control*, 12(5/6):549–552, 1968.

[6] S. Hariharan and P. Shankar. Evaluating the role of context in syntax directed compression of xml documents. In *Proceedings of the 2006 IEEE Data Compression Conference (DCC 2006)*, page 453, 2006.

[7] J. M. Hitchcock. *Effective Fractal Dimension: Foundations and Applications*. PhD thesis, Iowa State University, 2003.

[8] J. I. Lathrop and M. J. Strauss. A universal upper bound on the performance of the Lempel-Ziv algorithm on maliciously-constructed data. In B. Carpentieri, editor, *Compression and Complexity of Sequences '97*, pages 123–135. IEEE Computer Society Press, 1998.

[9] C. League and K. Eng. Type-based compression of xml data. In *Proceedings of the 2007 IEEE Data Compression Conference (DCC 2007)*, pages 272–282, 2007.

[10] A. Lempel and J. Ziv. Compression of individual sequences via variable rate coding. *IEEE Transaction on Information Theory*, 24:530–536, 1978.

[11] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[12] M. López-Valdés and E. Mayordomo. Dimension is compression. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 676–685. Springer-Verlag, 2005.

[13] E. Mayordomo. Effective fractal dimension in algorithmic information theory. In *New Computational Paradigms: Changing Conceptions of What is Computable*, pages 259–285. Springer-Verlag, 2008.

[14] D. Sculley and C. E. Brodley. Compression and machine learning: A new perspective on feature space vectors. In *Proceedings of the Data Compression Conference (DCC-2006)*, pages 332–341, 2006.